日　本　国　特　許　庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application: ２０００年　９月　７日

出願番号
Application Number: 特願２０００－２７０９２５

出願人
Applicant(s): 沖電気工業株式会社

２００１年　８月　３日

特許庁長官
Commissioner,
Japan Patent Office 及川耕造

出証番号　出証特２００１－３０６８１９５

【書類名】　特許願
【整理番号】　SA003564
【あて先】　特許庁長官殿
【国際特許分類】　G06F　9/46
【発明者】
【住所又は居所】　東京都港区虎ノ門１丁目７番１２号　沖電気工業株式会社内
【氏名】　内海　功朗
【特許出願人】
【識別番号】　000000295
【氏名又は名称】　沖電気工業株式会社
【代理人】
【識別番号】　100082050
【弁理士】
【氏名又は名称】　佐藤　幸男
【手数料の表示】
【予納台帳番号】　058104
【納付金額】　21,000円
【提出物件の目録】
【物件名】　明細書　1
【物件名】　図面　1
【物件名】　要約書　1
【包括委任状番号】　9100477
【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　タスクプログラム制御システム

【特許請求の範囲】

【請求項１】　複数のタスクの優先度と優先度別順位とを示す情報を記憶する記憶回路と、

前記記憶回路より、最も優先度の高いタスクの優先度を読み出す優先度読出回路と、

前記優先度読出回路で読み出した優先度の優先度別順位データを読み出す順位読出回路と、

前記優先度読出回路の値を保持する優先度レジスタと、

前記優先度読出回路で読み出した優先度内の順位の値を保持する順位レジスタと、

前記順位読出回路で読み出した優先度別順位データのうち、最も高い順位の値を前記順位レジスタに設定すると共に、当該順位レジスタの値が読み出された場合、そのタイミングで、当該読み出された値が、次に読み出す場合に最も低い順位となるよう設定する優先度別順位制御回路と、

前記優先度レジスタおよび順位レジスタの値を、次に実行するタスクのアドレスとして読み出し、そのタスクを実行し、実行後は当該タスクが有するタスク自身を継続するか中止するかの値に基づき前記記憶回路の情報の更新を行うタスク実行手段とを備えたことを特徴とするタスクプログラム制御システム。

【請求項２】　請求項１に記載のタスクプログラム制御システムにおいて、

任意のタスクのアドレスが優先度レジスタおよび順位レジスタから読み出された場合、記憶回路の当該タスクに該当する情報をクリアするよう構成したことを特徴とするタスクプログラム制御システム。

【請求項３】　請求項１または２に記載のタスクプログラム制御システムにおいて、

タスク実行手段からのリードアクセスの値をカウントするカウンタと、

特定の優先度に対応した所定値を有し、前記カウンタのカウント値が当該所定値を超えた場合に前記カウント値を出力するマスク回路と、

前記マスク回路からカウント値が出力された場合は、記憶回路より前記特定の優先度以下の優先度のうち最も優先度の高いタスクの優先度を読み出す特定優先度読出回路とを備えたことを特徴とするタスクプログラム制御システム。

【請求項４】　請求項１～３のいずれかに記載のタスクプログラム制御システムにおいて、

任意のタスクに対応したプログラムアドレスとデータアドレスとを示すプログラムテーブルとデータテーブルとを用意し、

タスク実行手段は、優先度レジスタおよび順位レジスタから任意のタスクのアドレスを読み出した場合は、前記プログラムテーブルとデータテーブルのアドレスに基づき、該当するタスクプログラムを実行し、かつ、データをアクセスするよう構成されたことを特徴とするタスクプログラム制御システム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、例えばネットワークプロセッサにおけるプログラム制御を行うタスクプログラム制御システムに関するものである。

【０００２】

【従来の技術】

従来、ハードウェアで行われていたネットワーク制御の処理をソフトウェアで行うネットワークプロセッサには、例えば、Lebel One(TM) IXP1200 Network Processor DataSheetに示されている技術があった。このネットワークを処理するプロセッサでは、四つのスレッド毎にレジスタバンクを持っているので、コンテキストの切替えはレジスタバンクの切替えで可能となる。但し、この構成ではレジスタバンクが４段しかないため、４より多いスレッドが実行される場合、コンテキストの退避が必要となり、多数のタスクを制御するのは困難である。

【０００３】

また、コンテキストスイッチではなく、タスクを分割しポーリングすることによって、タスクのコンテキストスイッチのサイクル数を削減する方法として、例えば、特開平１０－６３５１７号公報「リアルタイム情報処理およびその装置」

に示されている方法があった。

【０００４】

【発明が解決しようとする課題】

しかしながら、上記レジスタバンクによる切替えの方法は、特化したプロセッサによる方法であり、汎用プロセッサではできない。また、レジスタバンクが四つのスレッドに対応しているため、４よりも数多いタスクを並列に実行すると、再びコンテキストの退避、回復の問題が発生する。

また、特開平１０－６３５１７号公報に開示されている方法では、ポーリングのためにタスクを一定時間内に１回起動しなければならない、といった無駄なポーリング処理が必要となることや、タスクを分割しなければならないという問題点があった。

【０００５】

【課題を解決するための手段】

本発明は、前述の課題を解決するため次の構成を採用する。

〈構成１〉

複数のタスクの優先度と優先度別順位とを示す情報を記憶する記憶回路と、記憶回路より、最も優先度の高いタスクの優先度を読み出す優先度読出回路と、優先度読出回路で読み出した優先度の優先度別順位データを読み出す順位読出回路と、優先度読出回路の値を保持する優先度レジスタと、優先度読出回路で読み出した優先度内の順位の値を保持する順位レジスタと、順位読出回路で読み出した優先度別順位データのうち、最も高い順位の値を順位レジスタに設定すると共に、順位レジスタの値が読み出された場合、そのタイミングで、読み出された値が、次に読み出す場合に最も低い順位となるよう設定する優先度別順位制御回路と、優先度レジスタおよび順位レジスタの値を、次に実行するタスクのアドレスとして読み出し、そのタスクを実行し、実行後はタスクが有するタスク自身を継続するか中止するかの値に基づき記憶回路の情報の更新を行うタスク実行手段とを備えたことを特徴とするタスクプログラム制御システム。

【０００６】

〈構成２〉

構成１に記載のタスクプログラム制御システムにおいて、任意のタスクのアドレスが優先度レジスタおよび順位レジスタから読み出された場合、記憶回路のそのタスクに該当する情報をクリアするよう構成したことを特徴とするタスクプログラム制御システム。

【０００７】

〈構成３〉

構成１または２に記載のタスクプログラム制御システムにおいて、タスク実行手段からのリードアクセスの値をカウントするカウンタと、特定の優先度に対応した所定値を有し、カウンタのカウント値が所定値を超えた場合にカウント値を出力するマスク回路と、マスク回路からカウント値が出力された場合は、記憶回路より特定の優先度以下の優先度のうち最も優先度の高いタスクの優先度を読み出す特定優先度読出回路とを備えたことを特徴とするタスクプログラム制御システム。

【０００８】

〈構成４〉

構成１～３のいずれかに記載のタスクプログラム制御システムにおいて、任意のタスクに対応したプログラムアドレスとデータアドレスとを示すプログラムテーブルとデータテーブルとを用意し、タスク実行手段は、優先度レジスタおよび順位レジスタから任意のタスクのアドレスを読み出した場合は、プログラムテーブルとデータテーブルのアドレスに基づき、該当するタスクプログラムを実行し、かつ、データをアクセスするよう構成されたことを特徴とするタスクプログラム制御システム。

【０００９】

【発明の実施の形態】

以下、本発明の実施の形態を具体例を用いて詳細に説明する。

本発明は、コンテキストスイッチのサイクル数を削減するための、状態遷移で制御されたタスクプログラムに対して高速にタスクを切替えるようにしたシステムである。

【００１０】

《具体例１》

〈構成〉

図１は、本発明のタスクプログラム制御システムの具体例１を示す構成図であるが、これに先立ち、タスクプログラムの基本的な制御方法について説明する。

【００１１】

図２は、本発明の具体例１の全体的な構成図である。

図示の装置は、ＣＰＵ１００、タスクスケジューラ１１０、メインメモリ１２０、バス１３０からなる。ＣＰＵ１００は、タスクスケジューラ１１０の決定に従い、メインメモリ１２０に格納されている各種のプログラムを実行するプロセッサであり、本具体例では特定のＣＰＵは想定していない。

タスクスケジューラ１１０は、図１に示されている回路で構成され、ＣＰＵ１００に対して次に実行させるタスクの番号を生成するための機能を有している。

メインメモリ１２０は、各種のプログラムやデータを格納し、また、ＣＰＵ１００の作業領域として用いられるメモリであり、プログラムテーブル１２１、メインルーチン１２２、各種タスクプログラム１２３を格納している。プログラムテーブル１２１、メインルーチン１２２および各種タスクプログラム１２３については後述する。

また、バス１３０は、ＣＰＵ１００、タスクスケジューラ１１０、メインメモリ１２０を接続するシステムバスである。

【００１２】

先ず、ＣＰＵ１００で実行されるタスクのプログラムの構成と、そのプログラムを起動させるメインルーチン１２２の構成について説明する。

例えば、ネットワークでなされるプロトコル処理は、物理層以外の下位レイヤの処理においても、次のようなシーケンスとみなして処理が可能である。

図３は、典型的なプロトコルシーケンスの処理例の説明図である。

プロトコルの処理上、各シーケンスの処理内容は単純であるが、他のシーケンスと並列に動作し、通信しながら実行させることによって、複雑な制御が可能である。

図４は、図３のプロトコルの処理を関数で表した説明図である。

図中、proc0～2は、各処理の名札に相当する。また、recieve(chanel0,data);は通信路chanel0からデータdataを受けることを示す。send(chanel1,data);は通信路chanel1へデータdataを転送することを示す。return;は状態を開始状態に設定することを示す。

【００１３】

ところで、図３に示すシーケンス処理において、通常、そのシーケンスだけに閉じている処理に比べて、受信処理や送信処理はサイクル数がかかる。これは受信や送信の相手のシーケンサが用意できるまで、実行が待たされるためである。この場合、リアルタイムＯＳ上では、別のタスクを実行する制御が取られる。

【００１４】

図５は、本発明の具体例１で実行されるタスクプログラムの説明図であり、（ａ）はフローチャート、（ｂ）は、フローチャートの内容をＣ言語で記述したものである。

図３のように状態を持ったシーケンス処理は状態に応じた分岐命令と次の状態設定の形に書き換えできる。

ステップＳ１は、そのタスクの状態を保持する変数である。

ステップＳ２は、状態変数に応じた分岐命令である。

ステップＳ３は、状態に応じた処理内容で、構成は、ステップＳ３－１～ステップＳ３－４に分かれている。

即ち、ステップＳ３－１は、状態を表す数値で、ステップＳ２の分岐命令は、この値によって分岐する。

ステップＳ３－２は、状態に応じた処理である。

ステップＳ３－３は、次に実行される時の状態を設定する。

ステップＳ３－４は、リターン命令で、タスクから抜け出る。このときに、タスクスケジューラ１１０によって再度実行されるときは“１”を、待ち状態あるいは終了するときは“０”を返す。

【００１５】

図５（ｂ）中のコメント（／／）は、図５（ａ）中のステップＳ＊＊に対応している（＊＊は任意の数値を表す）。

図５（ｂ）中のcase1:にあるget(chane10,data)は、図４のrecieve(chane10,data)に対応する。違いはrecieve(chane10,data)が、dataがchane10に来るまで待つことに対して、get(chane10,data)は、既にchane10にあるデータdataをリードするだけでよく、ウエイトは発生しない。図５（ｂ）中の構成では、待つ処理を明示的に示さず、コメントS3-4のreturn(0)によって、処理をメインルーチン１２２に返すことにしている。

【００１６】

図６は、タスク呼出しを行うメインルーチン処理の説明図であり、（ａ）はフローチャート、（ｂ）は（ａ）の処理をＣ言語で示した説明図である。

先ず、ステップＳ１１で、特定のアドレスに割り付けられたタスクスケジューラアドレス（図６（ｂ）のMACRO_SCHEDULER_ADR）をレジスタ（図６（ｂ）のsp）に設定する。次に、ステップＳ１２で、タスクスケジューラ１１０から次に実行するタスク番号（図６（ｂ）のNUM）を取り出す。タスクスケジューラ１１０は、いつでも次に実行するタスク番号を生成する。

次に、ステップＳ１３で、そのタスク番号のプログラム（図６（ｂ）におけるfunc[NUM]()）を実行する。

そして、ステップＳ１４で、タスクプログラムの実行の返り値をタスク番号に対応した状態（図６（ｂ）のstate[NUM]）に反映させる。

タスクの状態は、次に説明するタスクスケジューラ１１０内のビットマトリクスに相当する。また、タスク番号は、ビットマトリクスのアドレスに対応している。そして、ステップＳ１２～ステップＳ１４を無限ループで実行する。

【００１７】

次に、図１に示されているタスクスケジューラ１１０の構成について説明する。

図１に示すシステムは、記憶回路であるマトリクス回路１、優先度読出回路である優先度プライオリティエンコーダ（ＰＥ）２、デコーダ（ＤＥＣＯＤＥ）３、ドライバ（ＤＲ）４、順位読出回路であるマルチプレクサ（ＭＵＸ）５、優先度別順位制御回路６（インバータ（ＩＶ）７、ＡＮＤ回路８，９、プライオリティエンコーダ（ＰＥ）１０，１１、ＯＲ回路１２、マルチプレクサ（ＭＵＸ）１

３、優先度別順位レジスタ１４、マルチプレクサ（ＭＵＸ）１５、デコーダ（ＤＥＣ）１６、減算器１７）、優先度レジスタ（ＰＲＩＲ）１８、順位レジスタ（ＯＤＲＲ）１９、タスク実行手段２０からなる。

【００１８】

マトリクス回路１は８×８ビットマトリクスであり、図面縦方向は優先度、横方向は同優先度での順位を表す。尚、上の方が優先度が高く、また、左の方が同一優先度での順位が高い。本具体例では、８通りの優先度で、各優先度で８通りの順位を持つ。但し、本発明はこのような優先度および順位の数に限定されるものではない。このマトリクス回路１では、各タスク毎に１ビットの状態を持つ。ここで、０は待ち状態または休止状態、１は実行可能状態である。優先度の数が大きく、また、同優先度で順位の高い、実行可能状態にあるビットに割り付けられたタスクがＣＰＵ１００で実行されるよう構成されている。

【００１９】

優先度プライオリティエンコーダ２は、各優先度毎に実行可能状態になっているビットの論理和（ＯＲ）を取り、その論理和に対してプライオリティエンコードして、実行可能状態にある最も高い優先度を検出する機能を有している。出力は３ビットで、優先度７の行にビットが立っていた場合出力は１１１となり、どの優先度のラインにもビットが立っていない場合出力は０００となる。

【００２０】

デコーダ３は、ＣＰＵ１００からのアクセス要求とアドレスから優先度レジスタ１８と順位レジスタ１９のデータをデータバスに出力するためのドライブ信号を生成する機能を有している。また、このドライブ信号は優先度別順位レジスタ１４の値を更新するのにも使用される。ドライバ４はデータバスへのドライバである。

マルチプレクサ５はマトリクス回路１から優先度プライオリティエンコーダ２が指定した優先度のラインビット（優先度別順位データ）を取り出すためのマルチプレクサである。

【００２１】

優先度別順位制御回路６は、マルチプレクサ５で読み出した優先度別順位デー

タのうち、最も高い順位の値を順位レジスタ１９に設定すると共に、この順位レジスタ１９の値が読み出された場合、そのタイミングで、読み出された値が、次に読み出す場合に最も低い順位となるよう設定する機能を有しており、インバータ７～減算器１７から構成されている。

インバータ７は、デコーダ１６の出力を反転する回路である。ＡＮＤ回路８は、インバータ７で反転したデコーダ１６の出力とマルチプレクサ５の出力との論理積（ＡＮＤ）を取る回路である。また、ＡＮＤ回路９は、デコーダ１６の出力とマルチプレクサ５の出力との論理積（ＡＮＤ）を取る回路である。

【００２２】

プライオリティエンコーダ１０，１１は、それぞれＡＮＤ回路８，９の出力をプライオリティエンコードして最も順位の高いビットを求めるためのプライオリティエンコーダである。即ち、プライオリティエンコーダ１０は、デコーダ１６から出力されるマスク信号に基づくビットのマスク部分で最も順位の高いビットを求めるプライオリティエンコーダであり、プライオリティエンコーダ１１は、マスクされていない部分で最も順位の高いビットを求めるプライオリティエンコーダである。

ＯＲ回路１２は、ＡＮＤ回路９の出力の論理和（ＯＲ）を取る回路である。ＡＮＤ回路９の出力に１ビットでも１が立っていた場合は１が出力されることになる。また、マルチプレクサ１３は、二つのプライオリティエンコーダ１０，１１からの出力を選択するマルチプレクサである。即ち、マスクされていない入力に対するプライオリティエンコーダ出力が常に優先して選択される。

【００２３】

優先度別順位レジスタ１４は、優先度別にそのレベルでのタスクが待ち状態に切り替わった時にその順位を保持するための回路である。尚、この保持タイミングは上述したようにデコーダ３からのドライブ信号がアサートされた時である。マルチプレクサ１５は、優先度プライオリティエンコーダ２出力に応じて優先度別順位レジスタ１４からの出力を選択するマルチプレクサである。デコーダ１６は、マルチプレクサ１５出力を３ビットから８ビットにデコードするためのデコーダである。即ち、デコーダ１６は、優先度別順位レジスタ１４に保持された順

位に基づくマスク信号を発生する。減算器１７は、順位レジスタ１９の値を１デクリメントさせる減算器である。

【００２４】

優先度レジスタ１８は、優先度プライオリティエンコーダ２の出力であるマトリクス回路１の優先度を保持するレジスタである。また、順位レジスタ１９は、マルチプレクサ１３の出力である優先度別順位の値を保持するレジスタである。

【００２５】

タスク実行手段２０は、ＣＰＵ１００がメインルーチン１２２を実行することで実現される機能であり、優先度レジスタ１８および順位レジスタ１９の値を、次に実行するタスクのアドレスとして読み出し、そのタスクを実行し、実行後はこのタスクが有するタスク自身を継続するか中止するかの値に基づきマトリクス回路１の情報の更新を行うものである。

【００２６】

〈動作〉

動作の一例として三つのタスクプログラムの動作する様子を示して、図５のタスクプログラム、図６のメインルーチンおよび図１の動作の説明を行う。

図７は、タスクの初期割付状況の説明図である。

図中、（ａ）は各タスクのアドレスを示している。図示のように、各タスクのアドレスは、６ビットの数値で表す。最初のｂはこれらのアドレスが２値表示であることを示す記号である。また、６ビットの“０”または“１”の中間に設けたアンダースコアは優先度と優先度別順位とを区別できるようにした表示で、実質的な意味はない。

【００２７】

図中、（ｂ）はマトリクス回路１におけるビット状態を示している。例えば、task0はアドレスとしてb110_101を持ち、８×８ビットマトリクス上では６行５列目に相当する場所に１が立っている。同じくtask1はアドレスとしてb011_110を持ち、８×８ビットマトリクス上では３行６列目に相当する場所に１が立っている。また、task2はアドレスとしてb011_011を持ち、８×８ビットマトリクス上では３行３列目に相当する場所に１が立っている。これらの設定は、図６（ｂ

）に示すメインルーチンプログラムが実行される前の初期設定で設定される。尚、マトリクス回路１において、b000-000は予約されている。

【００２８】

図８は、各タスクプログラムの説明図であり、（ａ）がtask0、（ｂ）がtask1、（ｃ）がtask2のプログラムを示している。

尚、各タスクプログラムの制御構造は図５（ｂ）と同様である。

【００２９】

以降、タスクスケジューラ１１０と三つのタスクプログラムがどのように動作するかを説明する。尚、ハードウェアとソフトウェアの説明を区別するため、ハードウェアの手順にはステップ＊＊＿Ｈで、ソフトウェアの手順にはステップ＊＊＿Ｓを表記するものとする（＊＊は任意の数値を表す）。

【００３０】

［ステップ１＿Ｈ］

マトリクス回路１から優先度プライオリティエンコーダ２は、データを取出す。図７に示した通り、１が立っている最大のラインは６行なので６（＝b110）を出力する。

［ステップ２＿Ｈ］

マルチプレクサ５より、優先度プライオリティエンコーダ２出力に従って６行目のビット列がマトリクス回路１より選択される。

［ステップ３＿Ｈ］

優先度別順位レジスタ１４からマルチプレクサ１５によって６行目の優先度順位を取り出す。通常、初期状態ではこれらの優先度順位はb111である。尚、b111は、十進法の７を示し、本具体例では最も優先度が高いことを表している。

【００３１】

［ステップ４＿Ｈ］

デコーダ１６でb111（＝７）に相当するパターンを８ビットのb11111111にデコードする。尚、デコードは、例えば、b000（＝０）はb00000001に、b001（＝１）は、b00000011のように行うものとする。

［ステップ５＿Ｈ］

このデコードされたデータは、ＡＮＤ回路８とＡＮＤ回路９でマルチプレクサ５から出力されたデータとＡＮＤ論理が取られる。ＡＮＤ回路８ではインバータ７によって反転されたデータとＡＮＤ論理が取られることになる。即ち、ＡＮＤ回路８では全てのビットがマスクされたデータ（b00000000）とマルチプレクサ５のデータとのＡＮＤ論理が取られ、ＡＮＤ回路９では全てのビットがマスクされていないデータ（b11111111）とマルチプレクサ５のデータとのＡＮＤ論理が取られる。その結果、ＡＮＤ回路８の出力はb00000000、ＡＮＤ回路９の出力はb00100000となる。

【００３２】

[ステップ６＿H]

プライオリティエンコーダ１０，１１の出力は、ＡＮＤ回路８，９の出力を通して、b000とb101である。また、ＯＲ回路１２でＡＮＤ回路９の出力においていずれかのビットが１であることが判明する。

[ステップ７＿H]

マルチプレクサ１３でＯＲ回路１２の値に従って、プライオリティエンコーダ１０，１１のいずれかの出力を選択する。ＯＲ回路１２の出力が１であるため、プライオリティエンコーダ１１の出力が選択され、その出力はb101である。

[ステップ８＿H]

以上の結果により、優先度レジスタ１８と順位レジスタ１９は、それぞれb110とb101が入る。これは、task0のアドレス値が上位と下位に分かれて入っていることに相当する。

【００３３】

[ステップ９＿S]

初期設定が終了して、図６（ｂ）の（Ｓ１１）で、ＣＰＵ１００は図示しないレジスタにタスクスケジューラアドレスを設定する。そして無限ループに入り、（Ｓ１２）でタスクスケジューラ１１０から上記ステップ８＿Hで得られたデータを取り出す。

【００３４】

[ステップ１０＿H]

デコーダ３で、ステップ９＿Ｓのリードアクセスをデコードして、ドライバ４を介してデータバス上に優先度レジスタ１８と順位レジスタ１９の値を出力する。

[ステップ１１＿Ｈ]

デコーダ３からのドライブ信号は、優先度別順位レジスタ１４の設定信号にもなっているので、順位レジスタ１９の値から１減少したデータb100が優先度レジスタ１８の相当するレベル、ここでは優先度６に対応するレジスタＬ６に設定される。

【００３５】

[ステップ１２＿Ｓ]

図６（ｂ）のメインルーチンでリードしてきたデータは変数NUMに設定される。そして、このNUMに対応する配列の関数を実行する（図６（ｂ）における（Ｓ１３）および図６（ａ）におけるステップＳ１３）。

【００３６】

図９は、メインメモリ１２０上でのプログラムテーブル１２１の割付状態を示す説明図である。

図示のように、メモリ上にプログラムテーブルとして配列funcが割り付けられ、task0，task1，task2の番号に相当するアドレスに、task0，task1，task2の実行開始アドレスが割り付けられている。従って、図６（ｂ）の（Ｓ１３）において、task0番地に分岐し、実行することに相当する。

【００３７】

[ステップ１３＿Ｓ]

図８（ａ）のtask0が実行され、状態変数sta0の値が初期値で0であることにより、case0に対応する命令が実行される（図８中の／／１３＿Ｓ）この部分で状態変数１に更新する（sta0＝1）。更に0が返される（return(0)）。

[ステップ１４＿Ｓ]

図６（ｂ）で、リターン値は配列stateのNUM番地に相当するところに0が書き込まれる。配列stateはマトリクス回路１の８×８ビットマトリクスの行と列に相当する。よって、マトリクス回路１でのNUM（＝b110_101）に対応するビット

が0にクリアされる。

［ステップ１５＿Ｈ］

マトリクス回路１の６行５列がクリアされたので、１が立っているのは３行の２ビットだけである。従って、優先度プライオリティエンコーダ２の出力は３となる。よって、マルチプレクサ５は３行目のビット列を取り出す。また、優先度別順位レジスタ１４の初期値はいずれもb111（＝７）なので、上記のステップ４＿Ｈ～ステップ８＿Ｈと同様にして、優先度レジスタ１８と順位レジスタ１９には、task1のアドレスであるb011とb110がそれぞれ入る。

【００３８】

［ステップ１６＿Ｓ］

メインルーチンの無限ループによって、再度、タスクスケジューラ１１０から次に実行すべき番号を取り出す（図６（ａ）、（ｂ）のステップＳ１２および／／（Ｓ１２））。

［ステップ１７＿Ｈ］

タスクスケジューラ１１０からのリードと共に、優先度別順位レジスタ１４の優先度３に対応するレジスタには順位レジスタ１９の値から１減少させた値101が入る。データバスにはb011110が出力される。

［ステップ１８＿Ｓ］

図６（ｂ）において、NUMにはb011110が設定され、関数の配列funcからアドレスb011110に相当するtask1を実行する（図６（ａ）、（ｂ）のステップＳ１３および／／（Ｓ１３））。

［ステップ１９＿Ｓ］

図８の（ｂ）に示すtask1が実行され、状態変数sta1の値が初期値で0であることにより、case0に対応する命令が実行される（図８の／／１９＿Ｓ）。この部分で状態変数を１に更新する（sta＝1）。更に１が返される（return(1)）。

［ステップ２０＿Ｓ］

図６（ｂ）において、リターン値は配列stateのNUM番地に相当するところに１が書き込まれる。配列stateはマトリクス回路１の行と列に相当する。よって、マトリクス回路１でのNUM（＝b011_101）に対応するビットが１に再設定される

。

【００３９】

［ステップ２１＿Ｈ］

マトリクス回路１の６行５列がクリアされたので、１が立っているのは３行の２ビットだけである。従って、優先度プライオリティエンコーダ２の出力は３となる。よって、マルチプレクサ５は３行目のビット列を取り出す。また、優先度別順位レジスタ１４の優先度３に相当するレジスタは101に更新されているので、デコーダ１６にはb00111111が出力される。

【００４０】

［ステップ２２＿Ｈ］

ＡＮＤ回路８，９でそれぞれデコーダ１６出力b00111111とマルチプレクサ５出力01001000の論理積を取ると、それぞれb01000000とb00001000が出力される。ＯＲ回路１２の出力が１となることにより、ＡＮＤ回路９の出力を入力とするプライオリティエンコーダ１１の出力が選択され、順位レジスタ１９にはb011が入る。

このような構成で、一度実行されたタスクは次に実行される場合、同優先度内では最後に実行されるようになる。即ち、６ビット目の順位のタスクが実行された後は、優先度別順位レジスタ１４の値がb101（＝５）となり、６、７ビット目のデータがマスクされるマスク信号がデコーダ１６から出力される。その結果、プライオリティエンコーダ１０の値はb110であり、プライオリティエンコーダ１１の値はb011であるが、ＡＮＤ回路９の出力に“１”が立っているため、プライオリティエンコーダ１１の出力が選択されることになる。

【００４１】

［ステップ２３＿Ｓ］

メインルーチンの無限ループによって、再度、タスクスケジューラ１１０から次に実行すべき番号を取り出す（図６（ａ）、（ｂ）におけるステップＳ１２および／／（Ｓ１２））。

［ステップ２４＿Ｈ］

タスクスケジューラ１１０からのリードと共に、優先度別順位レジスタ１４の

優先度３に対応するレジスタには順位レジスタ１９の値から１減少させた値010が入る。データバスにはb011011が出力される。

［ステップ２５＿Ｓ］

図６（ｂ）において、NUMにはb011011が設定され、関数の配列funcからアドレスb011011に相当するtask2を実行する（図６（ａ）、（ｂ）におけるステップＳ１３および／／（Ｓ１３））。

［ステップ２６＿Ｓ］

図８の（ｃ）に示すtask2が実行され、状態変数sta2の値が初期値で0であることにより、case0に対応する命令が実行される。“state[task0_id]”のtask0_idはtask0のアドレスb110101に相当し、１を書き込むということは、マトリクス回路１のb110（＝６）行、b101（＝５）列に１を書き込むことに相当する。

図８の（ｃ）中の／／２６＿Ｓの部分で状態変数１に更新する（sta2＝1）。更に１が返される（return(1)）。

【００４２】

［ステップ２７＿Ｓ］

図６（ｂ）において、リターン値は配列stateのNUM番地に相当するところに１が書き込まれる。配列stateはマトリクス回路１の行と列に相当する。よって、マトリクス回路１でのNUM（＝b011_011）に対応するビットが１に再設定される。

［ステップ２８＿Ｈ］

マトリクス回路１の６行目に１が立ったため、優先度プライオリティエンコーダ２はデータを取出し、図７に示した通り、１が立っている最大のラインは６行なので６を出力する。これ以降は、上述したステップ４＿Ｈ～ステップ８＿Ｈと同じ処理をたどる。

【００４３】

［ステップ２９＿Ｓ］

図６の（ｂ）のメインルーチンに示す無限ループによって、再度タスクスケジューラ１１０から次に実行すべき番号を取り出す（図６（ａ）、（ｂ）におけるステップＳ１２および／／（Ｓ１２））。NUMにはb110101が設定され、関数の配

列funcからアドレスb110101に相当するtask0を実行する（図６（ａ）、（ｂ）におけるステップＳ１３および／／（Ｓ１３））。

［ステップ３０＿Ｓ］

図８の（ａ）の状態変数sta0は1なので、case1以下の命令が実行される。状態変数sta0は0に設定し、return(0)で０を返す。

［ステップ３１＿Ｈ］

マトリクス回路１の６行５列がクリアされたので、１が立っているのは３行の２ビットだけである。従って優先度プライオリティエンコーダ２の出力は３となる。よって、マルチプレクサ５は３行目のビット列を取り出す。また、優先度別順位レジスタ１４の優先度３のレジスタ値はb010（＝2）なので、今度は、ＡＮＤ回路８，９の出力は、それぞれ、b10010000とb00000000となる。従って、ＯＲ回路１２の出力は０となり、マルチプレクサ１３の出力はプライオリティエンコーダ１０の出力b110で、これはtask1のアドレスの下位ビットである。よって、task1が実行されることになる。

［ステップ３２＿Ｓ］

以降、task1が起動し、続いてtask2が起動する。

【００４４】

図１０は、このようなタスクの実行順序を示す説明図である。

このように、マトリクス回路１の優先度のビット指定に従ってタスクが交互に実行される。また、スケジュール部分をハードウェアで、実行起動部分をソフトウェアでやっているので、どのようなＣＰＵ１００でも適用が可能である。

【００４５】

〈効果〉

以上のように、具体例１によれば、マトリクス回路１や優先度別順位制御回路６等からなるタスクスケジューラ１１０で、次に実行するタスクを決定し、かつ、各タスクは、自身を継続するか中止するかの情報を持ち、タスクを実行した場合は、この情報に基づいてマトリクス回路１の状態を更新するようにしたので、タスク切替えを高速に行うことができる。例えば、本具体例で示したＣ言語のプログラムおよび各タスクの状態名札への分岐部分はアセンブラプログラムでそれ

ぞれ５命令程度である。よって、実行中のタスクプログラムから次に優先順位の高いタスクプログラムへはデータアクセス、分岐命令によるロスを考慮しても２０サイクル以内で可能となる。汎用プロセッサにおいて、コンテキストスイッチが最低でも１００サイクル程度必要とされることを考えると、極めて高速な処理方法である。

また、本具体例ではＣＰＵ１００の構造に対して限定がなく、どのようなＣＰＵあるいはＤＳＰでも適用可能である。

【００４６】

《具体例２》

〈構成〉

図１１は、具体例２の全体構成図である。

上述した具体例１ではＣＰＵ１００が一つの場合であったのに対し、具体例２では複数のＣＰＵ１００（１００－１、１００－２）が同じバス１３０上に接続されている場合である。この場合のタスクスケジューラ１１０ａの構成は次のようになっている。

【００４７】

図１２は、具体例２におけるタスクスケジューラ１１０ａの要部構成図である。

基本的な構成は具体例１と同様であるが、タスクスケジューラ１１０ａをリードする、即ち、優先度レジスタ１８、順位レジスタ１９（図１２ではこれらの図示は省略している）を読む制御信号によって、そのアドレスデータに基づいてクリアするようになっている。他の各構成は図１に示した構成と同様であるため、ここでの説明は省略する。

【００４８】

〈動作〉

具体例１のタスクの例を用いて説明する。

先ず、ＣＰＵ１００－１がバス権を取得してタスクスケジューラ１１０ａにアクセスすると、具体例１と同様にtask0のアドレス値が得られる。ＣＰＵ１００－１はそのままtask0を実行する。ここで、具体例２では、優先度レジスタ１８

および順位レジスタ１９の値を読み出すと同時にtask0のアドレス値に対応するビットがクリアされる。

次に、ＣＰＵ１００－１がtask0を実行中、ＣＰＵ１００－２がタスクスケジューラ１１０ａにアクセスしたとすると、task0のビットはクリアされているので、task1のアドレス値が読み出され、task1が実行される。

【００４９】

このように、最も優先度が高いタスクに対応したタスクアドレスを読み出す度に、マトリクス回路１上のそのタスクアドレスに対応するビットをクリアすることで、複数のＣＰＵ１００が同一のタスクを実行するということがなくなり、複数のＣＰＵ１００であっても、優先順位別に別々のタスクを実行することができる。

【００５０】

〈効果〉

以上のように具体例２によれば、任意のタスクのアドレスが読み出された場合は、マトリクス回路１におけるそのタスクのアドレス値をクリアするようにしたので、複数のＣＰＵが存在した場合でも、重複することなくタスクを実行することができる。

【００５１】

《具体例３》

複数のタスクにおいて、優先度の高いタスクが多く割り付けられている場合、優先度の低いタスクが全く実行されなくなるといった事態が生じる。これは一般のリアルタイムＯＳ上でも発生している。これを解決するため、具体例３では、低い優先度のタスクでも周期が低い形で実行できるようにしている。

【００５２】

〈構成〉

図１３は、具体例３におけるタスクスケジューラの要部構成図である。

図では、具体例１と異なっている部分を示しており、マトリクス回路１および優先度プライオリティエンコーダ２は具体例１と同様である。具体例３では、更に、カウンタ（ＣＯＵＮＴＥＲ）２１、マスク回路（ＭＳＫ）２２、特定優先度

読出回路２３（ＡＮＤ回路２４、プライオリティエンコーダ２５、ＯＲ回路２６、マルチプレクサ２７）が追加されている。他の各構成は具体例１と同様であるため、その図示および説明は省略する。

【００５３】

カウンタ２１はリードアクセスによってカウントアップするカウンタである。また、ＯＲ回路２５の出力が“１”となってリードアクセスが行われると“０”にクリアされる。この例では、図面の上から下に向かってカウンタ値が増えるものとする。マスク回路２２は、カウンタ２１の出力のうち、あるビットの範囲だけマスクするマスク回路である。即ち、特定の優先度に対応した所定値を有し、カウンタ２１の値が所定値を超えた場合にそのカウント値を出力するよう構成されている。つまり、優先度が高いものは比較的実行する可能性が高いので、それをマスクして、優先度の低いものが実行できるようにさせるための回路である。特定優先度読出回路２３は、マスク回路２２からカウント値が出力された場合は、マトリクス回路１より特定の優先度のうち最も優先度の高いタスクの優先度を読み出す機能を有しており、ＡＮＤ回路２４～マルチプレクサ２７で構成されている。

【００５４】

ＡＮＤ回路２４は、マトリクス回路１の優先度出力とマスク回路２２出力とのＡＮＤ演算を行うＡＮＤ回路である。プライオリティエンコーダ２５は、ＡＮＤ回路２４の出力のうち最も優先度の高い優先度を取り出すものであり、優先度プライオリティエンコーダ２と同様３ビットの信号をＯＲ回路２６およびマルチプレクサ２７に出力する。ＯＲ回路２６は、プライオリティエンコーダ２５出力に、１であるビットがあるかどうかを判定し、あった場合は“１”を出力するＯＲ回路である。また、ＯＲ回路２６の出力はカウンタ２１のクリアイネーブルに接続されている。マルチプレクサ２７は優先度プライオリティエンコーダ２またはプライオリティエンコーダ２５のいずれかの出力を選択するセレクタであり、ＯＲ回路２６の出力が１であった場合にプライオリティエンコーダ２５の出力を選択するよう構成されている。

尚、図示省略しているが、具体例３のマルチプレクサ５は、マルチプレクサ２

７の出力が与えられ、この出力に基づいて優先度別順位を読み出すよう構成されている。

【００５５】

〈動作〉

図１４は、動作の一例を示すための説明図である。

マトリクス回路１には図示のようにビットが割り当てられているとする。このうち、６行目に割り当てられているタスクばかり実行していて、２行目に割り当てられたタスクは実行がされたことがないとする。

【００５６】

このような状態で、優先度プライオリティエンコーダ２の出力は６である。ここで、カウンタ２１の値がb00100000になったとする（図では、上位がマトリクス回路１のビットマトリクスの０行目、下位が７行目に対応する）。また、マスク回路２２では、上位３ビットのみ１とし、それ以外の下位は０にマスクされるよう設定されているとする。よって、マスク回路２２の出力はb00100000となる。

ＡＮＤ回路２４は、マトリクス回路１のビット出力b10110010とマスク回路２２出力b00100000とをＡＮＤ演算し、b00100000を出力する。その結果、プライオリティエンコーダ２５はb010（＝２）を出力する。これにより、ＯＲ回路２６の出力が１となり、マルチプレクサ２７はプライオリティエンコーダ２５の出力b010を選択して出力する。

【００５７】

即ち、カウンタ２１が６４回リードアクセスになったときには、優先的に優先度２のタスクが起動されることになる。また、ＯＲ回路２６の出力により次のタスクのリード時にはカウンタ２１がクリアされる。

また、上記例では、優先度２のタスクを優先的に起動する場合について説明したが、これ以外の優先度でもマスク回路２２のマスクビットの設定によって任意の優先度のタスクを起動させることができる。

【００５８】

〈効果〉

以上のように、具体例３によれば、リードアクセスの値をカウントするカウンタ２１と、このカウンタ２１出力を、特定の優先度に対応した値でマスクするマスク回路２２と、特定の優先度に対応したカウント値になった場合は、マトリクス回路１から、その優先度以下で最も優先度の高い優先度を読み出す特定優先度読出回路２３を設けたので、具体例１の効果に加えて、優先度の低いところに割り付けられたタスクでも周期的に選択されるので、優先度の低いタスクでも起動させることができるという効果がある。

【００５９】

《具体例４》

具体例４では、同じデータ構成であるが異なったデータに対して同じ処理を行う場合に効果が上がるようにしたものである。例えば、このような例としてはＡＴＭセルデータの処理がある。本具体例は主としてソフトウェア構成にかかわるものである。

【００６０】

〈構成〉

図１５は、具体例４の全体構成図である。

図の装置は、具体例１、２と比べてメインメモリ１２０内の構成が異なっている。メインメモリ１２０内には、プログラムテーブル１２１に加えてデータテーブル１２４が設けられ、かつ、メインルーチン１２２ａと各種タスクプログラム１２３を有している。

【００６１】

図１６は、プログラムテーブル１２１とデータテーブル１２４の構成説明図である。

具体例１の説明でも示した通り、タスクスケジューラ１１０は次に実行するタスク番号をＣＰＵ１００に渡し、ＣＰＵ１００はメインメモリ１２０上のプログラムテーブル１２１から分岐するプログラム先頭アドレスをアクセスして、そのアドレスに分岐することにより該当するタスクを実行していた。

これと同じく、次に実行するタスク番号からデータテーブル１２４をアクセスしてやり、そのデータに対して処理する構成をとる。図１６の（ａ）に示したプ

ログラムテーブル１２１では、タスクスケジューラ１１０が取る得るタスク番号に対応したアドレスにそのプログラム先頭アドレスが設定されている。また、（ｂ）に示したデータテーブル１２４では、同様のアドレス割付でデータがテーブルに設定されている。

【００６２】

これにより、タスク実行手段２０の機能を実現するメインルーチン１２２ａでは、マトリクス回路１より特定のタスク番号を読み出した場合、プログラムテーブル１２１とデータテーブル１２４のアドレスに基づき、そのプログラムを実行し、かつ、データにアクセスするよう構成されている。他の各構成は具体例１と同様であるため、ここでの説明は省略する。

【００６３】

〈動作〉

図１７は、具体例４で実行されるタスクプログラムの構造を示す説明図であり、（ａ）はそのフローチャート、（ｂ）は、フローチャートの内容をＣ言語で記述したものである。

図１７に示す具体例４のタスクプログラムと、図５で示した具体例１のタスクプログラムとの違いは、具体例４のタスクプログラムでは、状態変数の宣言（図５におけるステップＳ１）がないことである。これは、状態変数が関数の引数として渡されるデータの一要素となっているためである。

また、図１７（ｂ）のプログラム例と図５（ｂ）のプログラム例との違いは、関数の引数としてデータテーブル１２４のアドレス値（struct xxx*a）が渡されることと、状態変数がデータ構造体の一要素（a.state）として持つことである。尚、“struct xxx*a”はxxxという構造体を持つ変数aのアドレス値を表している。状態変数の更新は、データ構造体の一要素の更新が（a.state = ）で示されている。

【００６４】

図１８は、タスク呼出しを行うメインルーチン処理の説明図であり、（ａ）はそのフローチャート、（ｂ）はその処理をＣ言語で示した説明図である。

図１８（ａ）と図６（ａ）との違いは、ステップＳ１３ａのタスク番号に対応

したデータの取り出し処理、即ち、データテーブル１２４からタスク番号に対応したデータアドレスの取り出しが入るだけである。他のステップＳ１１ａ、Ｓ１２ａ、Ｓ１４ａ、Ｓ１５ａは、図６（ａ）におけるステップＳ１１、Ｓ１２、Ｓ１４と同様の処理である。

また、図１８（ｂ）のプログラム例と、図６（ｂ）のプログラム例の違いは、配列関数funcの引数に*data[NUM]が入っているだけである。これは、データテーブル１２４の配列data[]からタスク番号NUMのデータアドレスを取り出すことに相当する。

以上、このようなタスク番号に対して、プログラムテーブル１２１とデータテーブル１２４とを分けてやれば、異なったタスク番号において、同じプログラムで異なったデータに対する処理が可能となる。

【００６５】

図１９は、このような処理を行う場合のプログラムテーブル１２１とデータテーブル１２４の一例を示す説明図である。

図示のように、異なったタスク番号（b111100とb111010）において、同じプログラム（func1）で異なったデータ（data0,data1）に対する処理を行うことが可能となる。

【００６６】

〈効果〉

以上のように、具体例４によれば、任意のタスク番号に対応したプログラムアドレスとデータアドレスとを示すプログラムテーブル１２１とデータテーブル１２４とを設け、マトリクス回路１から特定のタスク番号を読み出した場合は、これらテーブルに示すアドレスに基づきタスクプログラムを実行し、かつ、データアクセスを行うようにしたので、例えば、同じデータ構成であるが異なったデータに対して同じ処理を行うような場合、データ個々にプログラムを設定する必要がなく、プログラムサイズが小さくて済む。これは、特にオブジェクト指向プログラムのようにデータとプログラムが一つの構造となったものに対して有効である。

【図面の簡単な説明】

【図１】

本発明のタスクプログラム制御システムの具体例１を示す構成図である。

【図２】

本発明の具体例１の全体的な構成図である。

【図３】

典型的なプロトコルシーケンスの処理例の説明図である。

【図４】

図３のプロトコルの処理を関数で表した説明図である。

【図５】

本発明の具体例１で実行されるタスクプログラムの説明図である。

【図６】

タスク呼出しを行うメインルーチン処理の説明図である。

【図７】

タスクの初期割付状況の説明図である。

【図８】

各タスクプログラムの説明図である。

【図９】

メインメモリ上でのプログラムテーブルの割付状態を示す説明図である。

【図１０】

タスクの実行順序を示す説明図である。

【図１１】

具体例２の全体構成図である。

【図１２】

具体例２におけるタスクスケジューラの要部構成図である。

【図１３】

具体例３におけるタスクスケジューラの要部構成図である。

【図１４】

具体例３の動作の一例を示すための説明図である。

【図１５】

具体例４の全体構成図である。

【図１６】

具体例４におけるプログラムテーブルとデータテーブルの構成説明図である。

【図１７】

具体例４で実行されるタスクプログラムの構造を示す説明図である。

【図１８】

タスク呼出しを行うメインルーチン処理の説明図である。

【図１９】

具体例４におけるプログラムテーブルとデータテーブルの一例を示す説明図である。

【符号の説明】

１　マトリクス回路（記憶回路）

２　優先度プライオリティエンコーダ（優先度読出回路）

５　マルチプレクサ（順位読出回路）

６　優先度別順位制御回路

１８　優先度レジスタ

１９　順位レジスタ

２０　タスク実行手段

２１　カウンタ

２２　マスク回路

２３　特定優先度読出回路

１００、１００－１、１００－２　ＣＰＵ

１１０、１１０ａ　タスクスケジューラ

１２０　メインメモリ

１２１　プログラムテーブル

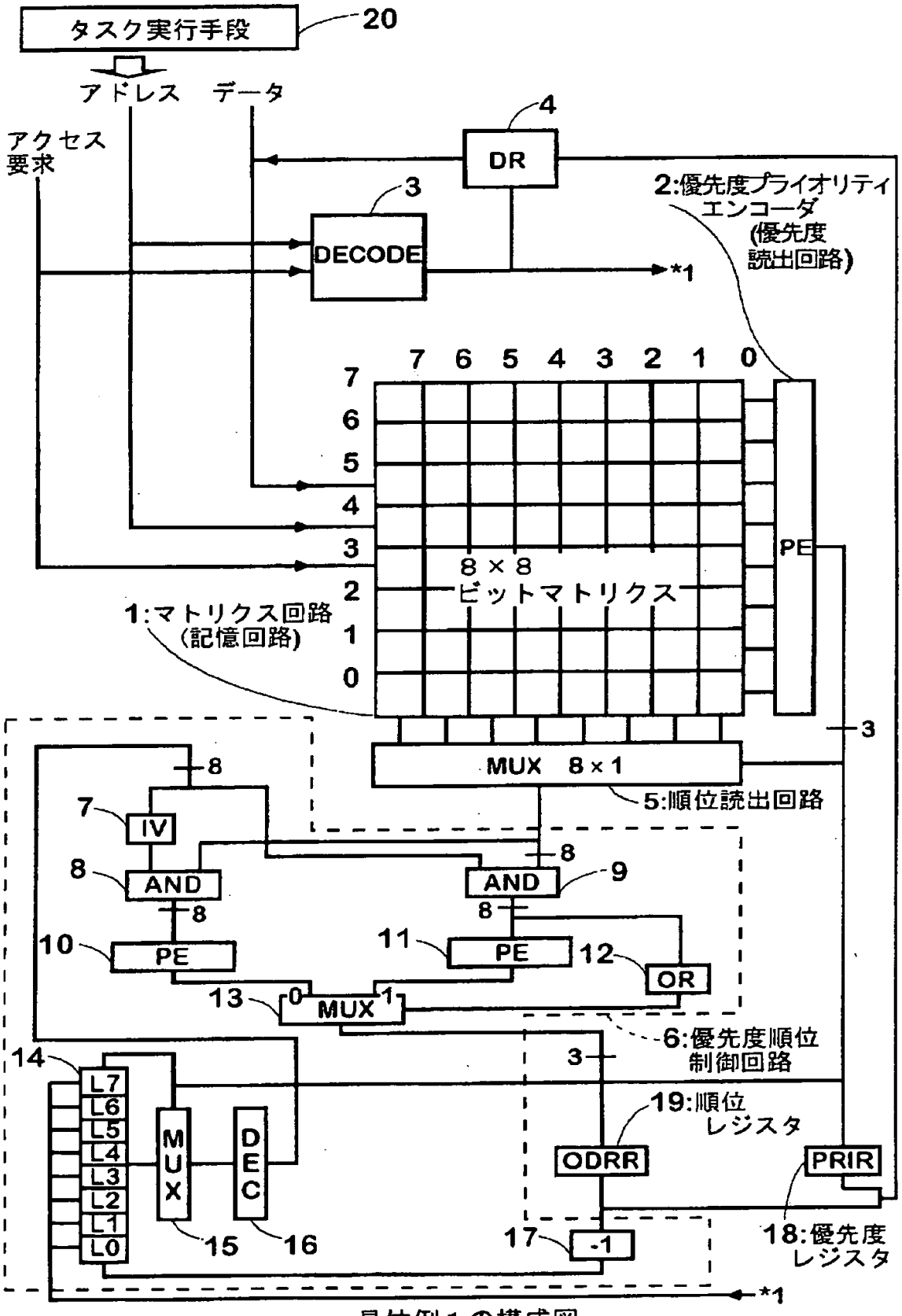
１２２、１２２ａ　メインルーチン

１２３　各種タスクプログラム

１２４　データテーブル

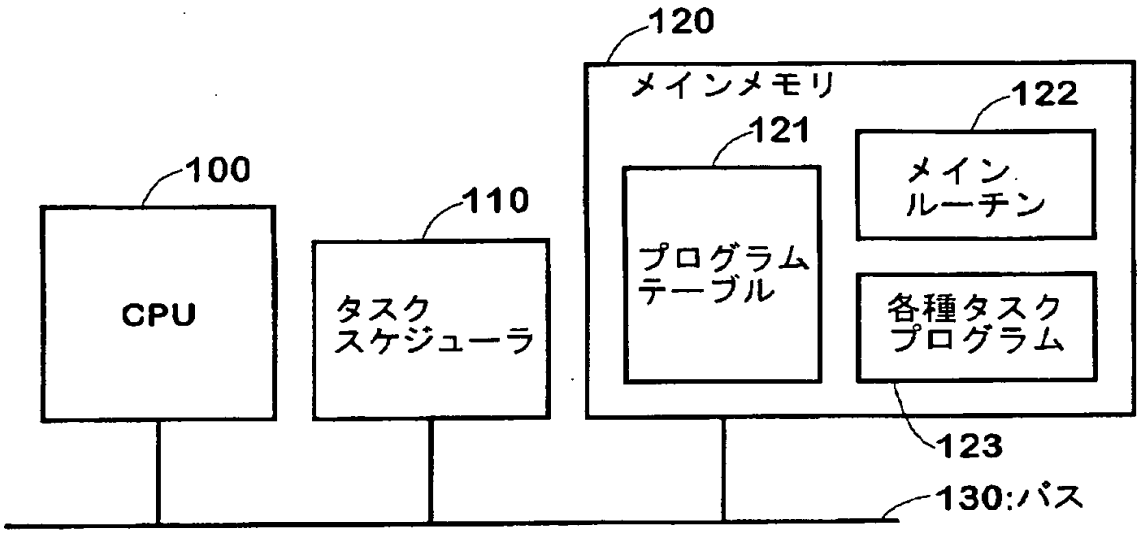
【書類名】図面

【図１】


タスク実行手段
20
アドレス
データ
アクセス
要求
DR
4
DECODE
3
*1
2:優先度プライオリティ
エンコーダ
(優先度
読出回路)
7 6 5 4 3 2 1 0
8×8
ビットマトリクス
1:マトリクス回路
(記憶回路)
PE
MUX 8×1
5:順位読出回路
IV
AND
AND
9
PE
PE
OR
MUX
6:優先度順位
制御回路
L7
L6
L5
L4
L3
L2
L1
L0
MUX
DEC
15
16
19:順位
レジスタ
ODRR
PRIR
17
-1
18:優先度
レジスタ

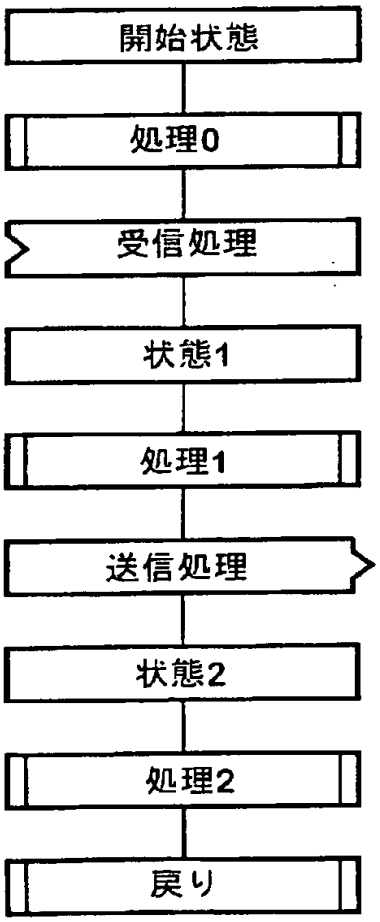
具体例１の構成図

【図２】



具体例１の全体構成図

【図３】



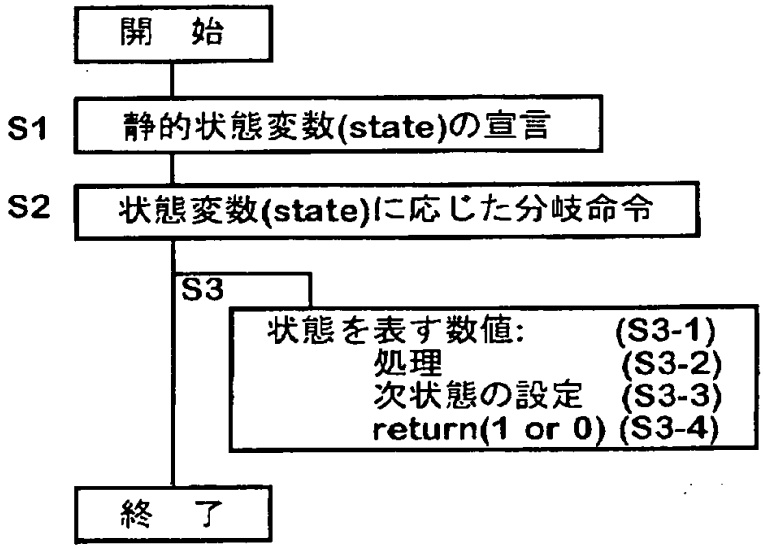
典型的なプロトコルシーケンスの処理例の説明図

【図４】

```
func()
{
    proc0:
        処理内容0;
    recieve(chanel0, data);
    proc1:
        処理内容1;
    send(chanel0, data);
    proc2:
        処理内容2;
    return;
}
```

関数表示例の説明図

【図５】



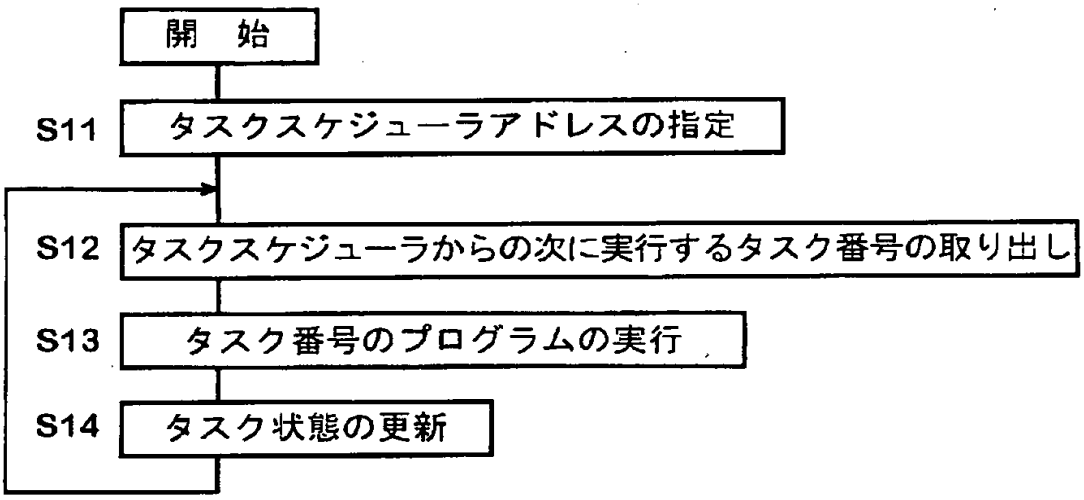
(a)

```
static int state;  // S1
func()
{
    switch(state&0x3) { // S2
    //S3
    case0:   //S3-1
        処理内容0;  //  S3-2
        state=1;    //  S3-3
        return(0);  //  S3-4
    case1:
        get(chanel0,data);
        処理内容1;
        send(chanel0, data);
        state=2;
        return(0);
    case2:
        処理内容2;
        state=0;
        return(0);
    defaults:
        state=0;
        return(0);
  }
}
```

(b)

具体例１で実行されるタスクプログラムの説明図

【図６】



(a)

```
int. *sp;
sp= TASK_SCHEDULER_ADR;  // S11

while(1) {
     NUM= *sp;                        // S12
     state[NUM]=func[NUM]();    // S13&S14
     }
```

(b)

メインルーチン処理の説明図

【図７】

task0 b110_101
task1 b011_110
task2 b011_011

(a)各タスクのアドレス

	7	6	5	4	3	2	1	0
7	0	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
3	0	1	0	0	1	0	0	0
2	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

(b)８×８ビットマトリクス状況

タスクの初期割付状況の説明図

【図８】

```
static int sta0;

int task0()
{
    switch(sta0&0x1) {
    case0:                    //13_S
        sta0=1;
        return(0);
    case1:                    //30_S
        sta0=0;
        return(0);
    }
}
```

(a)task0の内容

```
static int sta1;

int task1()
{
    switch(sta1&0x1) {
    case0:                    //19_S
        sta1=1;
        return(1);
    case1:
        sta1=0;
        return(0);
    }
}
```

(b)task1の内容

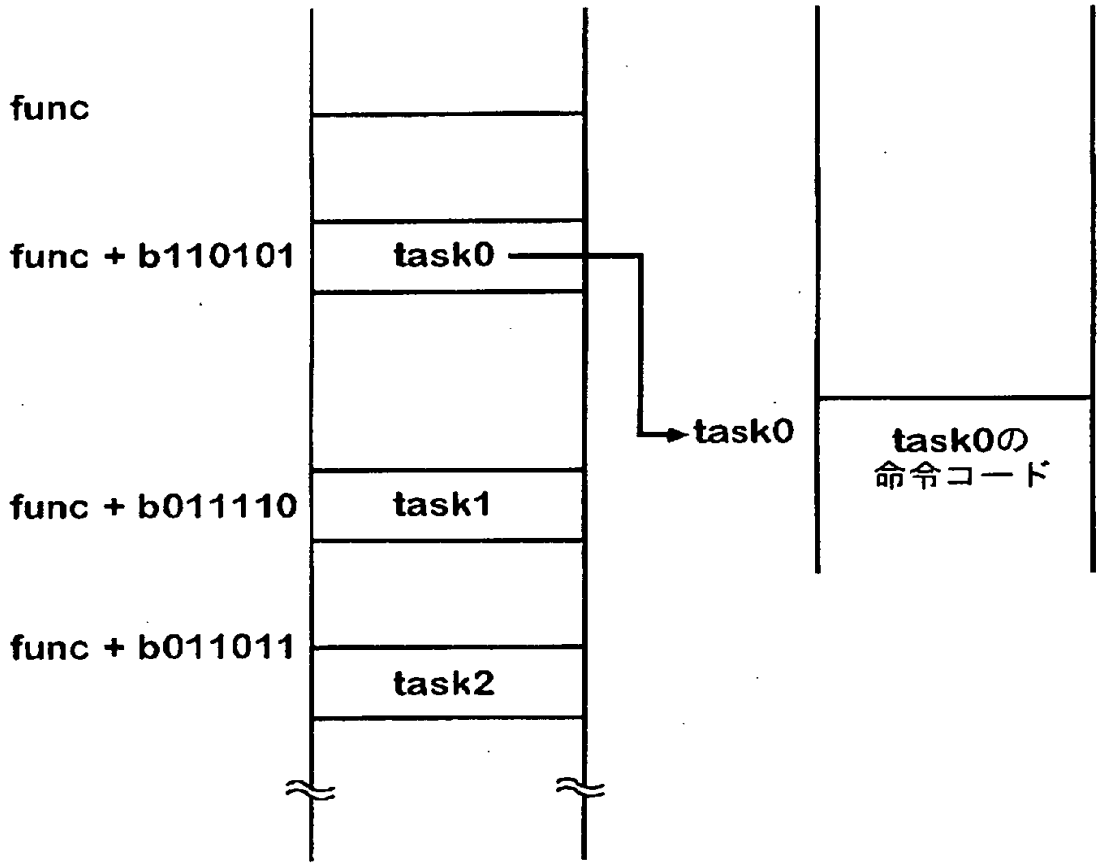
```
static int sta2;

int task2()
{
    switch(sta2&0x1) {
    case0:                    //26_S
        state(task0_id)=1;
        sta2=1;
        return(1);
    case1:
        sta2=0;
        return(0);
    }
}
```

(c)task2の内容

各タスクプログラムの説明図

【図９】

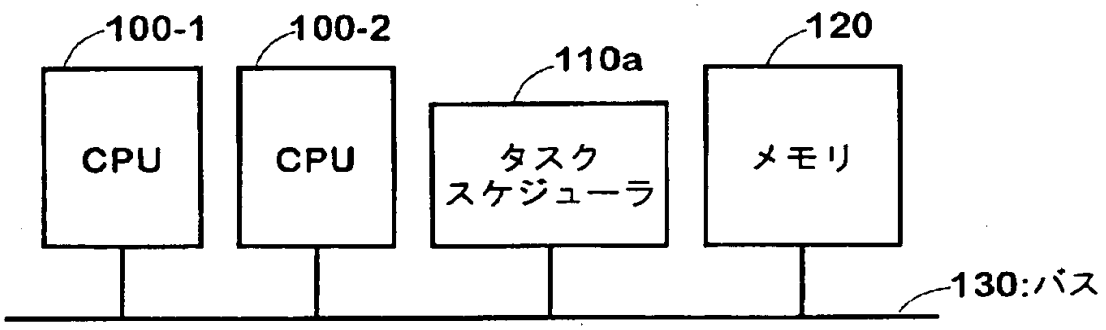


メモリ上でのプログラムテーブル割付状態の説明図

【図１０】

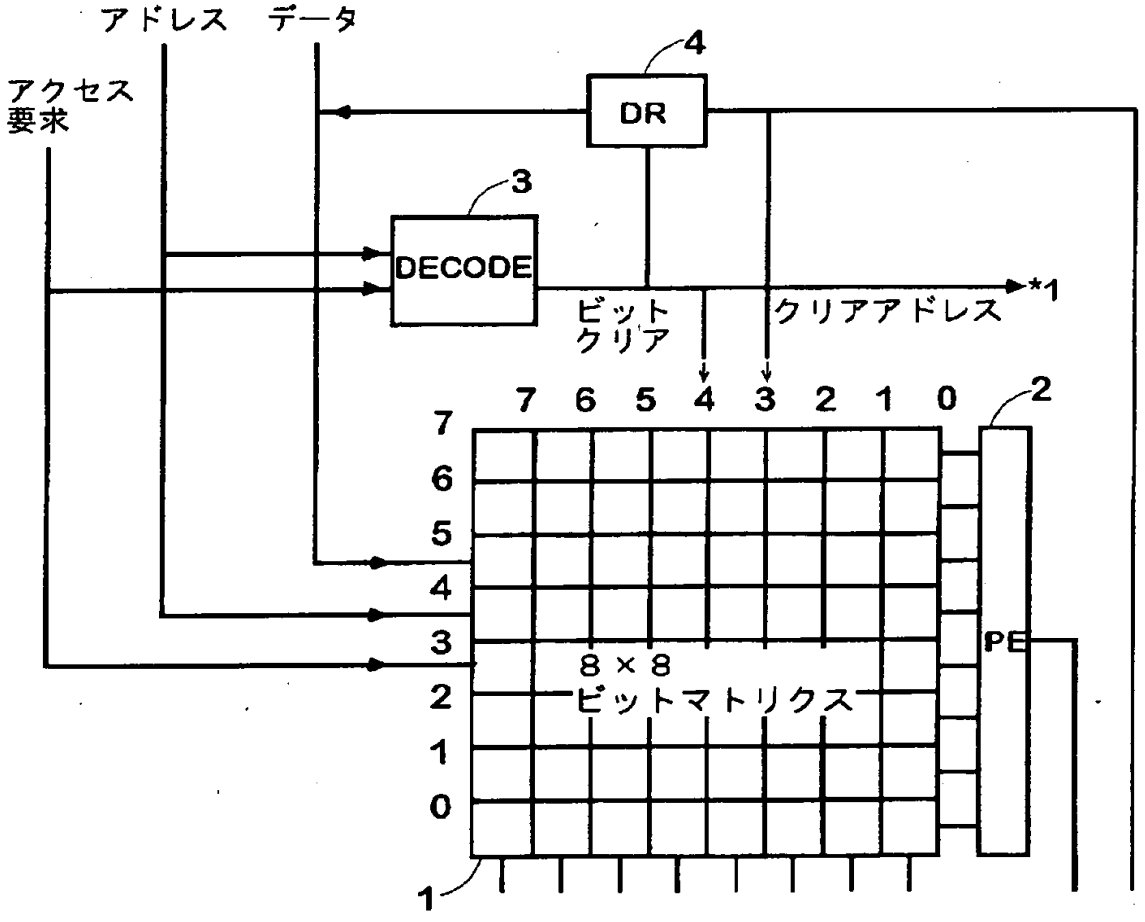
タスクの実行順序の説明図

【図１１】



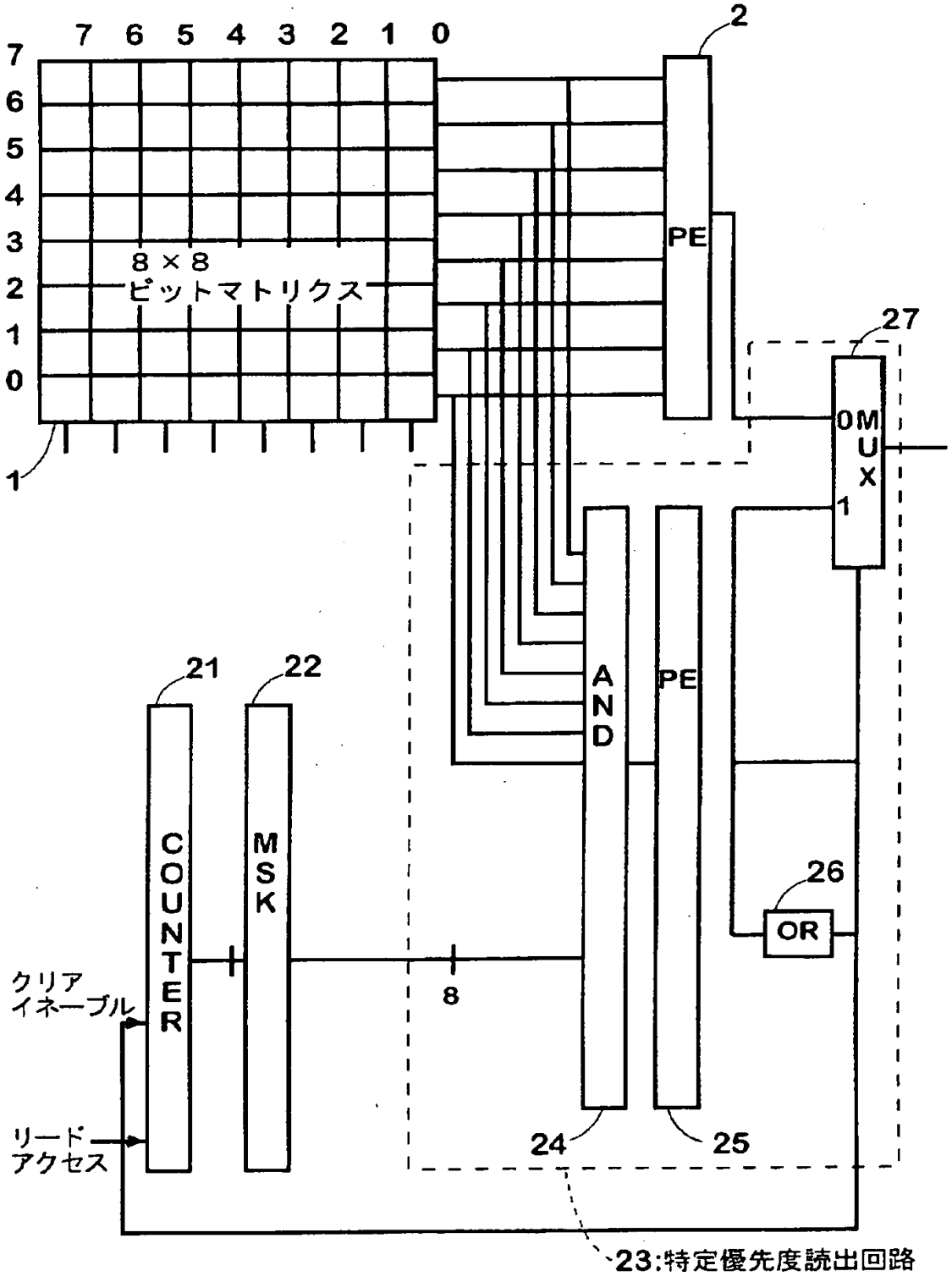
具体例２の全体構成図

【図１２】



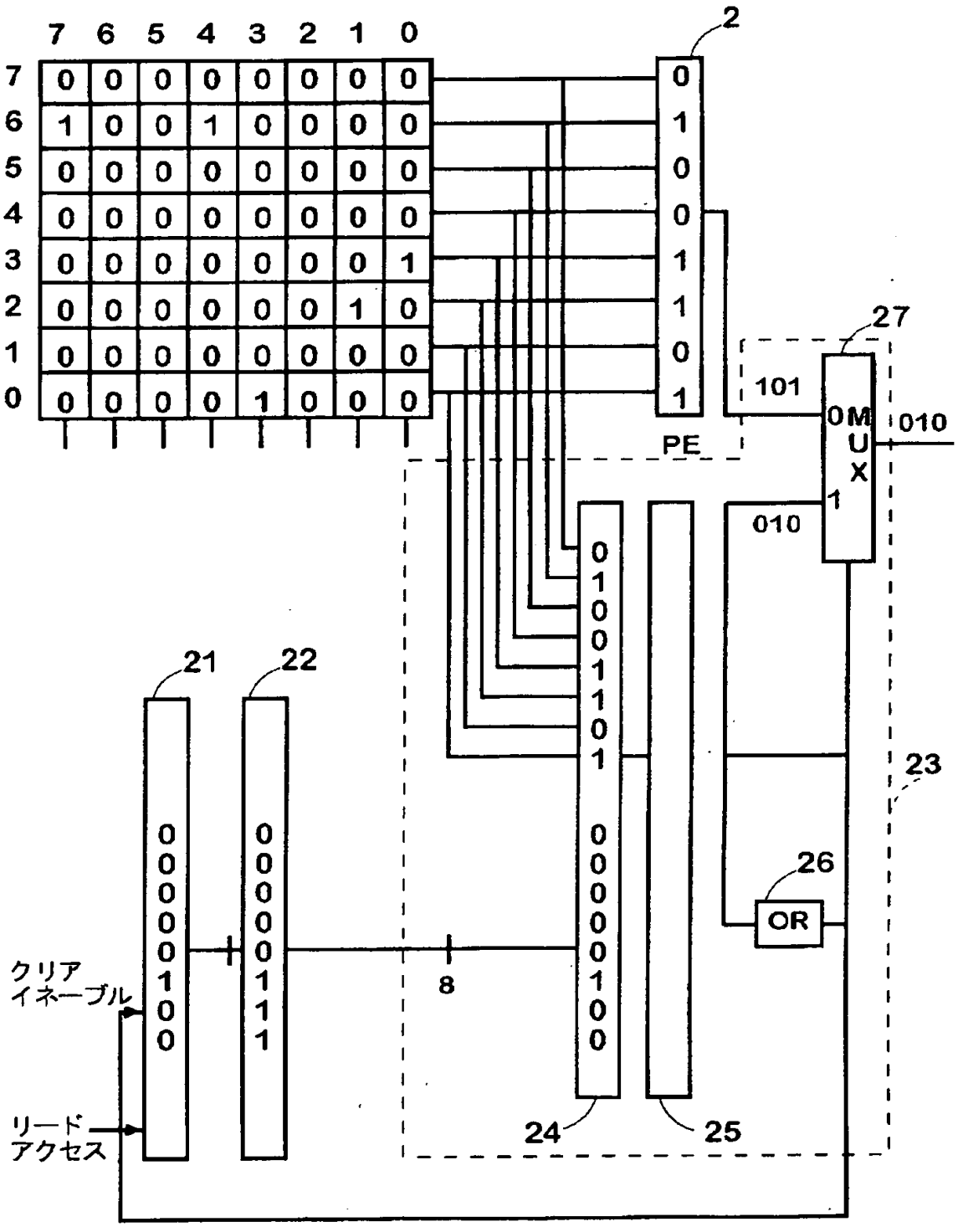
具体例２のタスクスケジューラの要部構成図

【図１３】



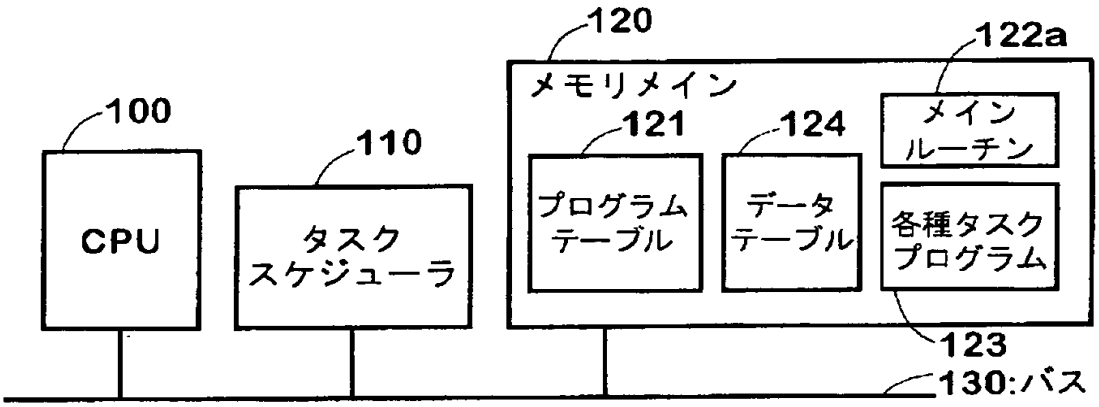
具体例３のタスクスケジューラの要部構成図

【図１４】



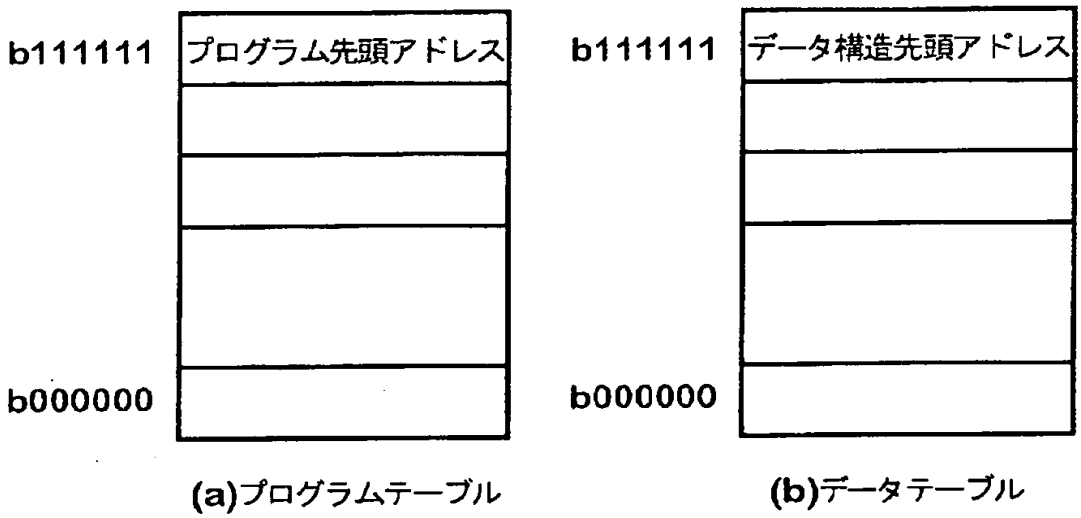
具体例３の動作の一例の説明図

【図１５】



具体例４の全体構成図

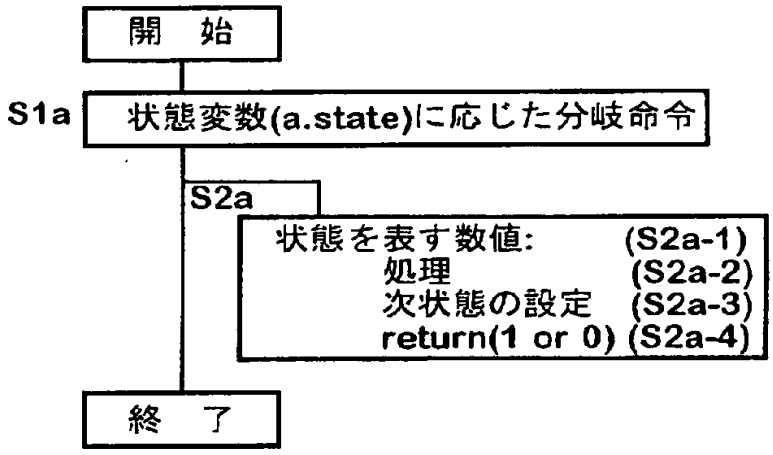
【図１６】



具体例４のプログラムテーブルとデータテーブルの説明図

出証特２００１－３０６８１９５

【図１７】



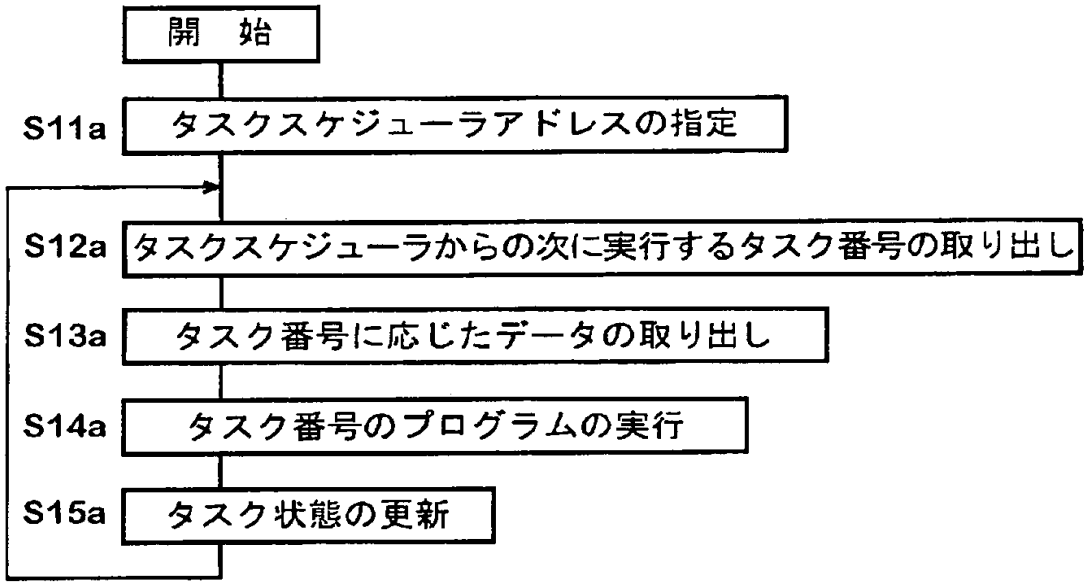
(a)

```
int.  func(struct xxx*a)
{
     switch(a.state&0x3) { // S1a
     //S2a
     case0:    //S2a-1
            処理内容0;   //   S2a-2
            state=1;     //   S2a-3
            return(0);   //   S2a-4
     case1:
            get(chanel0,data);
            処理内容1;
            send(chanel0, data);
            a.state=2;
            return(0);
     case2:
           処理内容2;
           a.state=0;
            return(0);
     defaults:
            a.state=0;
            return(0);
    }
}
```

(b)

具体例４で実行されるタスクプログラムの説明図

【図１８】



(a)

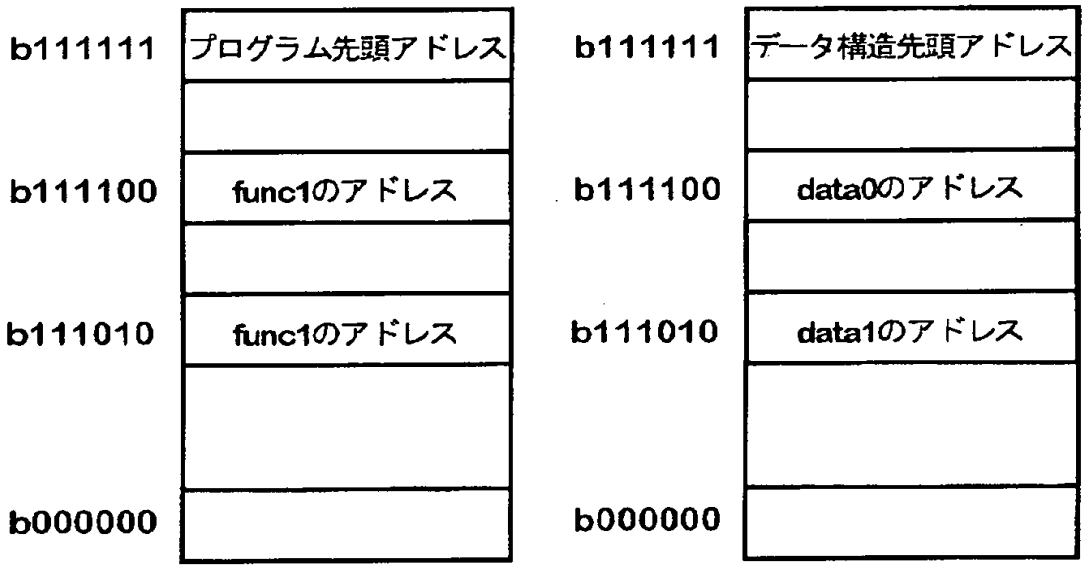
```
int. *sp;
int. *data[];
sp= TASK_SCHEDULER_ADR;  // S11a

while(1) {
    NUM= *sp;                          // S12a
    state[NUM]=func[NUM](data[NUM]);  // S14a&15a&S13a
    }
```

(b)

具体例４のメインルーチン処理の説明図

【図１９】



具体例４のプログラムテーブルとデータテーブルの一例の説明図

【書類名】　　要約書

【要約】

【課題】　複数のタスクの切替えを高速に行う。

【解決手段】　マトリクス回路１には複数のタスクの優先度と優先度別順位の情報が記憶されている。優先度プライオリティエンコーダ２は、マトリクス回路１から最も優先度の高い値を取り出す。この値は優先度レジスタ１８に保持される。優先度別順位制御回路６は、優先度別順位のうち最も順位の高い値を順位レジスタ１９に設定し、かつ、この値が次に読み出す場合は最も低い順位となるよう設定する。タスク実行手段２０は、次に実行するタスクのアドレスを優先度レジスタ１８および順位レジスタ１９から読み出し、そのタスクを実行し、実行後は、タスクの状態に基づきマトリクス回路１の情報を更新する。

【選択図】　　図１

認定・付加情報

特許出願の番号	特願２０００－２７０９２５
受付番号	５００００１１４１４９３
書類名	特許願
担当官	第七担当上席　　００９６
作成日	平成１２年　９月　８日

＜認定情報・付加情報＞

【提出日】　平成12年 9月 7日

次頁無

出　願　人　履　歴　情　報

識別番号　　〔０００００２９５〕

１．変更年月日　　１９９０年　８月２２日
［変更理由］　新規登録
住　所　　東京都港区虎ノ門１丁目７番１２号
氏　名　　沖電気工業株式会社